1 Problème du plus court chemin

Dans tout ce paragraphe, on considère un graphe connexe valué simple $G = (S,A)$, orienté ou non.

1.1 Présentation du problème

Le problème du plus court chemin est un problème classique de la théorie des graphes valués : il s'agit de déterminer un chemin entre deux sommets donnés de façon à ce que la somme des coûts des arêtes/arcs du chemin soit minimale. Ce problème a de multiples applications, comme le calcul d'itinéraires dans un GPS, l'optimisation de ressources, les tables de routage en informatique, etc.

Ce problème se présente sous différentes formes, dont la plupart est entièrement résolue à l'aide d'algorithmes très efficaces.

- ⋆ Sous sa forme la plus simple, l'origine du chemin est *unique* et tous les coûts sont *positifs*. Plusieurs algorithmes de résolution sont connus, le plus classique étant l'*algorithme de Dijkstra* que nous étudions en détail ci-dessous.
- ⋆ D'autres formes classiques :
  - origine du chemin unique, coûts quelconques et absence de circuits de coûts négatifs (algorithme de Bellman-Ford) ;
  - origine du chemin unique et coûts quelconques (problème non résolu) ;
  - origine et terminaison du chemin quelconque, coûts quelconques et absence de circuits de coûts négatifs (algorithmes de Floyd-Warshall et Johnson).

  On peut également ajouter des contraintes additionnelles, comme le passage obligatoire par certains sommets, la précédence entre sommets, des contraintes cumulatives sur la capacité, etc.

1.2 Algorithme de Dijkstra

◅ **Données de l'algorithme.**

- ⋆ un graphe valué simple $G = (S,A)$ à coûts *positifs* ;
- ⋆ un sommet initial $s \in S$, appelé *la source*.

◅ **Idée générale de l'algorithme.**

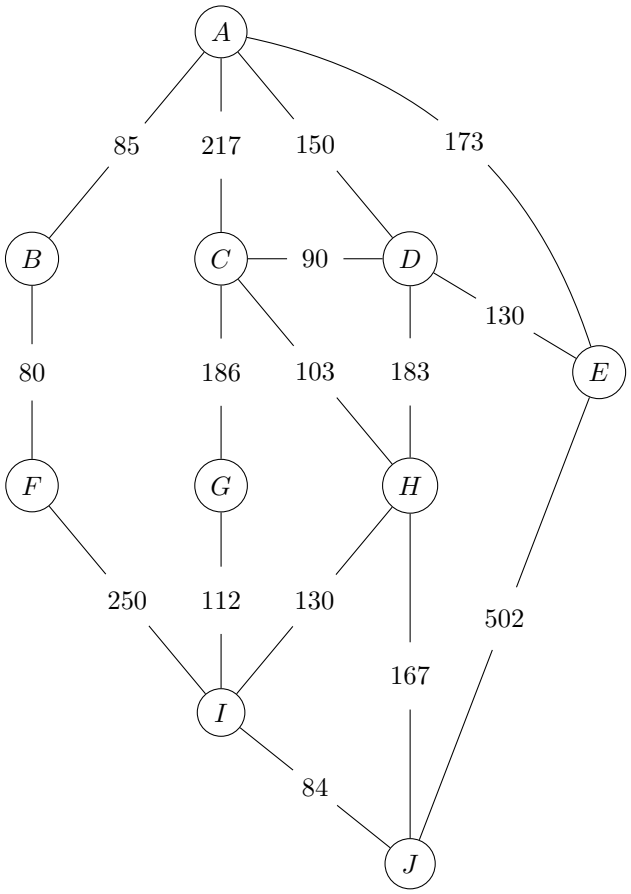
- ⋆ On calcule progressivement les plus courtes distances d_i de s vers chaque sommet i du graphe G.
- ⋆ A chaque itération, on traite le sommet i non encore traité de plus petite valeur d_i (car on est sûr avec les coûts positifs qu'il ne pourra plus être amélioré).
- ⋆ On crée progressivement une arborescence des plus courts chemins de racine s par la structure `père`.

◅ **Détail de l'algorithme.** L'algorithme de Dijkstra est basé sur l'algorithme du parcours en largeur.

```
1  fonction dijkstra(G,s)
2     SVus ← {s}
3     pour tout sommet i ∈ S-SVus
4        si (s,i) ∈ A alors
5           d_i ← C(s,i)
6           i.père ← s
7        sinon
8           d_i ← ∞
9        fin si
10    fin pour
11    tant que SVus ≠ S
12       déterminer t dans S-SVus tel que d_t soit le minimum des d_i pour i dans S-SVus
13       ajouter t à SVus
14       pour tout sommet k de S-SVus adjacent à t
15          si d_k ⩾ d_t + C(t,k) alors
16             d_k ← d_t + C(t,k)
17             k.père ← t
18          fin si
19       fin pour
20    fin tant que
21    renvoyer l'arbre de racine s
22 fin fonction
```

Les lignes 2 à 10 correspondent à la phase d'initialisation et les lignes 11 à 20 à la boucle principale de l'algorithme.

◅ **Un exemple pas à pas.** On considère le graphe suivant :



Les tableaux ci-dessous illustrent l'exécution de l'algorithme de Dijkstra en prenant comme source le sommet A :

- l'étape 1 est la phase d'initialisation de l'algorithme : les distances des sommets adjacents au sommet A sont indiquées dans le tableau et les distances des sommets non adjacents au sommet A sont mises à ∞ ;
- dans les étapes 2 à 10, on complète le tableau par l'ajout d'une ligne correspondant au sommet le plus proche de A parmi les sommets non encore vus. Chaque distance indiquée sur cette ligne est obtenue en additionnant la distance entre le sommet A et le sommet que l'on traite avec la distance entre le sommet que l'on traite et ses sommets adjacents. Si la distance obtenue est supérieure à celle que l'on a déjà depuis le sommet A, on la barre et si cette distance est inférieure à celle que l'on a déjà depuis le sommet A, on la garde et on barre la précédente.

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
A		85	217	150	173	∞	∞	∞	∞	∞

(a) Etape 1 : initialisation avec le sommet A

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
A		85	217	150	173	~~∞~~	∞	∞	∞	∞
B						165				

(b) Etape 2 : le sommet B est le sommet non vu le plus proche de A

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
A		85	217	150	173	~~∞~~	∞	~~∞~~	∞	∞
B						165				
D			~~240~~		~~280~~			333		

(c) Etape 3 : le sommet D est le sommet non vu le plus proche de A

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
A		85	217	150	173	~~∞~~	∞	~~∞~~	~~∞~~	∞
B						165				
D			~~240~~		~~280~~			333		
F									415	

(d) Etape 4 : le sommet F est le sommet non vu le plus proche de A

	A	B	C	D	E	F	G	H	I	J
A		85	217	150	173	~~∞~~	∞	~~∞~~	~~∞~~	~~∞~~
B						165				
D			~~240~~		~~280~~			333		
F									415	
E										675

(e) Etape 5 : le sommet E est le sommet non vu le plus proche de A

	A	B	C	D	E	F	G	H	I	J
A		85	217	150	173	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~
B						165				
D			~~240~~		~~280~~			~~333~~		
F									415	
E										675
C							403	320		

(f) Etape 6 : le sommet C est le sommet non vu le plus proche de A

	A	B	C	D	E	F	G	H	I	J
A		85	217	150	173	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~
B						165				
D			~~240~~		~~280~~			~~333~~		
F									415	
E										~~675~~
C							403	320		
H									~~450~~	487

(g) Etape 7 : le sommet H est le sommet non vu le plus proche de A

	A	B	C	D	E	F	G	H	I	J
A		85	217	150	173	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~
B						165				
D			~~240~~		~~280~~			~~333~~		
F									415	
E										~~675~~
C							403	320		
H									~~450~~	487
G									~~515~~	

(h) Etape 8 : le sommet G est le sommet non vu le plus proche de A

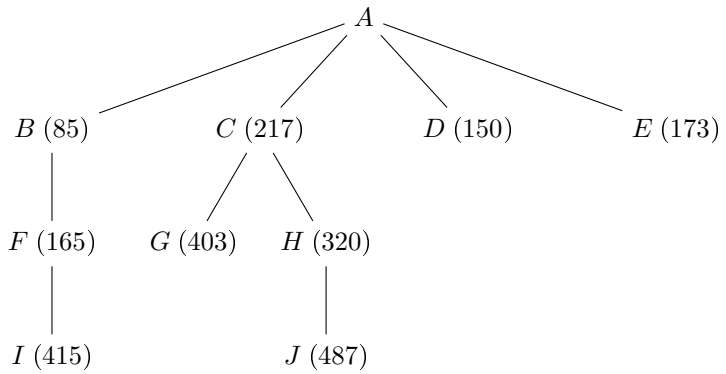
	A	B	C	D	E	F	G	H	I	J
A		85	217	150	173	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~
B						165				
D			~~240~~		~~280~~			~~333~~		
F									415	
E										~~675~~
C							403	320		
H									~~450~~	487
G									~~515~~	
I										~~499~~

(i) Etape 9 : le sommet I est le sommet non vu le plus proche de A

	A	B	C	D	E	F	G	H	I	J
A		85	217	150	173	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~
B						165				
D			~~240~~		~~280~~			~~333~~		
F									415	
E										~~675~~
C							403	320		
H									~~450~~	487
G									~~515~~	
I										~~499~~
J										

(j) Etape 10 : le sommet J est le sommet non vu le plus proche de A

A la fin de l'algorithme, on obtient l'arbre des plus courts chemins de racine A suivant :



Exercice 1.1 Suivant l'algorithme de Dijkstra, quel est le plus court chemin entre A et J ?

⊲ **Implémentation et complexité.** Si l'utilisation de l'algorithme « à la main » est très simple pour des graphes raisonnables, l'implémentation de l'algorithme de Dijkstra est en revanche extrêmement complexe, notamment l'implémentation de la recherche du minimum. Avec une bonne implémentation, on peut montrer que la complexité de l'algorithme est $O(\text{card}(A)+\text{card}(S)\log_2(\text{card}(S)))$.

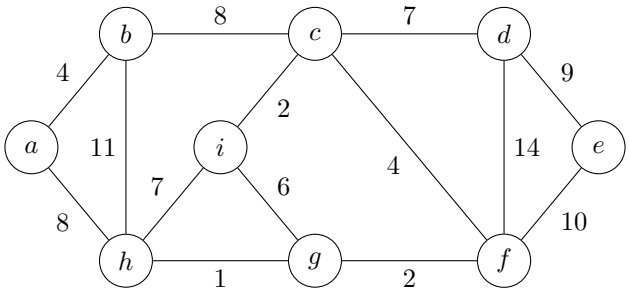
2 Arbre couvrant minimal

Dans tout ce paragraphe, on considère un graphe connexe non orienté $G = (S,A)$, valué et simple.

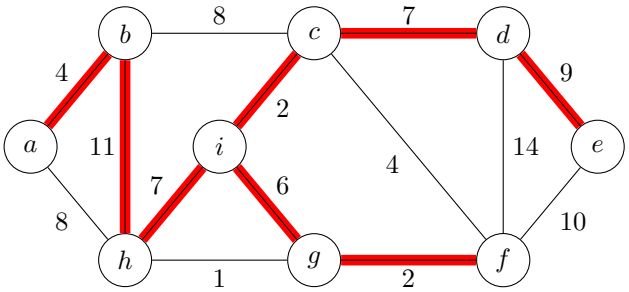
2.1 Présentation du problème

Le problème de l'arbre couvrant minimal est un autre problème classique de la théorie des graphes valués : il s'agit de déterminer un *arbre couvrant*, c'est-à-dire un arbre connectant ensemble tous les sommets du graphe, dont la somme des coûts des arêtes est *minimale*, c'est-à-dire inférieure ou égale à celle de n'importe quel autre arbre couvrant. Autrement dit, un *arbre couvrant minimal* représente la façon la plus économique de connecter ensemble tous les sommets du graphe.

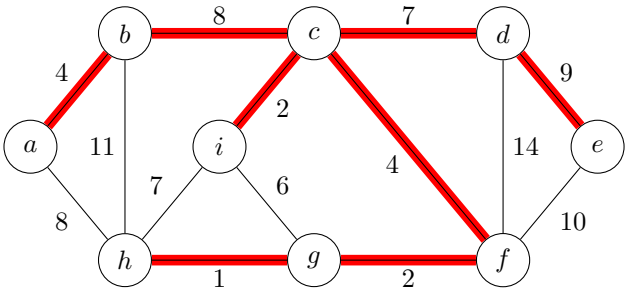
Exemple 2.1 Considérons le graphe suivant :



Les figures 3a et 3b ci-dessous donnent deux arbres couvrants de ce graphe, celui de la figure 3b étant minimal :



(a) Arbre couvrant de poids 48



(b) Arbre couvrant de poids 37

FIGURE 3 – Exemples d'arbres couvrants

Remarque 2.2 Un arbre couvrant minimal n'est pas unique. Par exemple, dans l'arbre de la figure 3b, on peut remplacer l'arête (b,c) par l'arête (a,h) et l'arbre obtenu est encore un arbre couvrant minimal.

Théorème 2.3 *Le graphe G a toujours au moins un arbre couvrant minimal.*

Démonstration. La connexité du graphe G implique qu'il admet toujours au moins un arbre couvrant. De plus, comme G a un nombre fini de sommets, alors il y a nécessairement un nombre fini d'arbres couvrants, d'où un nombre fini de coûts pour l'ensemble des arbres couvrants et donc au moins un arbre couvrant dont le coût est minimal. ■

Il existe de nombreux algorithmes pour résoudre le problème de l'arbre couvrant minimal. Dans ce chapitre, nous nous intéresserons aux algorithmes classiques de Kruskal (1956) et de Prim (1957).

Le problème de l'arbre couvrant minimal a de multiples applications dans des domaines très variés, comme les réseaux téléphoniques, les réseaux électriques, les réseaux de distributions, le partitionnement de données, le traitement d'image, la conception de circuits électroniques (par exemple : l'interconnexion de broches de composants pour les rendre électriquement équivalents), etc.

2.2 Un algorithme générique

Les algorithmes de détermination d'un arbre couvrant minimal utilisent une *stratégie gloutonne* pour faire pousser l'arbre recherché arête par arête. Techniquement, il s'agit de gérer un ensemble d'arêtes $\mathcal{E}$ tout en conservant la propriété suivante :

« *avant chaque itération, $\mathcal{E}$ est un sous-ensemble d'un certain arbre couvrant minimal* ».

Ainsi, à chaque étape, on doit déterminer une arête $(u,v) \in A$ qui peut être ajoutée à l'ensemble $\mathcal{E}$ tout en respectant cette propriété, c'est-à-dire que $\mathcal{E} \cup \{(u,v)\}$ est encore un sous-ensemble d'un arbre couvrant minimal. Une telle arête est appelée *arête sûre pour $\mathcal{E}$*, car on peut l'ajouter à l'ensemble $\mathcal{E}$ sans détruire la propriété recherchée. On a alors l'algorithme générique suivant :

```
fonction arbre_couvrant_minimal_generique(G)
  E ← ∅
  tant que E ne forme pas un arbre couvrant de G
    trouver une arête (u,v) qui est sûre pour E
    E ← E ∪ {(u,v)}
  fin tant que
  renvoyer E
fin fonction
```

Les algorithmes de Kruskal et de Prim que nous détaillons ci-dessous utilisent une règle spécifique pour déterminer une arête sûre.

2.3 Algorithme de Kruskal

◅ **Donnée de l'algorithme.**

- un graphe connexe non orienté $G = (S,A)$, valué et simple.

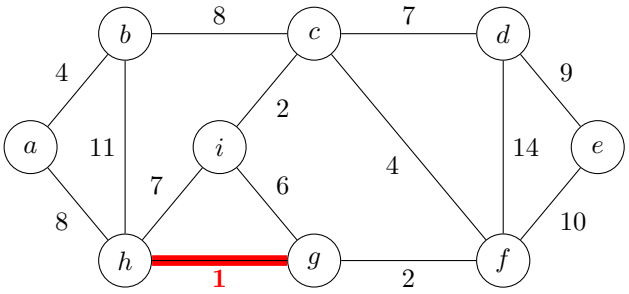
◅ **Idée générale de l'algorithme.**

L'algorithme de Kruskal est basé sur le fait qu'un arbre est *un graphe sans cycle*. Son principe est le suivant :

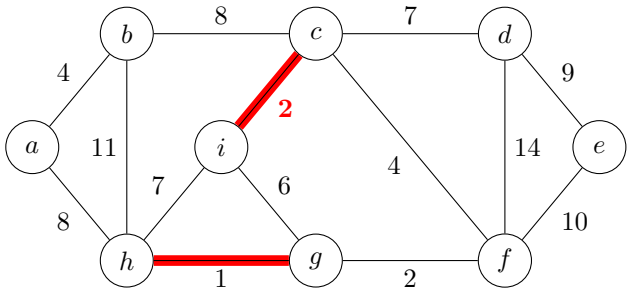
- Trier les arêtes par ordre croissant de coûts.
- Dans cette liste, choisir une arête de coût minimal qui ne forme pas un cycle avec les arêtes déjà retenues.
- Recommencer jusqu'à ce que l'on ait examiné toutes les arêtes.

Avec une bonne implémentation, la complexité de l'algorithme de Kruskal est en $O(\text{card}(A)\log_2(\text{card}(S)))$.

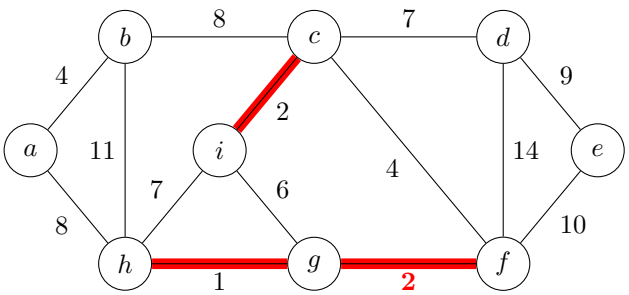
◅ **Un exemple pas à pas.** Les figures ci-dessous illustrent l'exécution de l'algorithme de Kruskal sur le graphe de l'exemple 2.1.



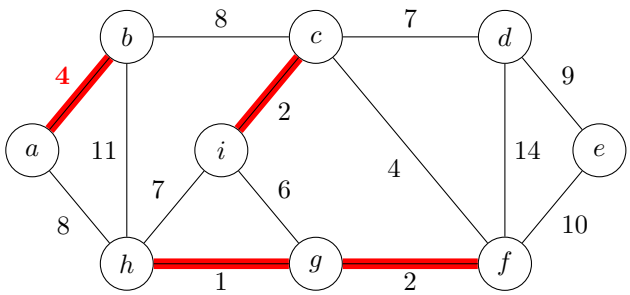
(a) Etape 1 : (h,g) de coût minimal et ne forme pas un cycle



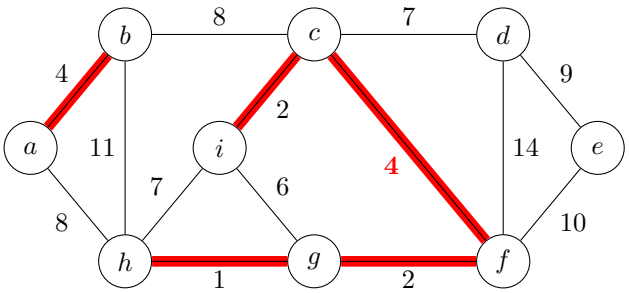
(b) Etape 2 : (i,c) de coût minimal et ne forme pas un cycle



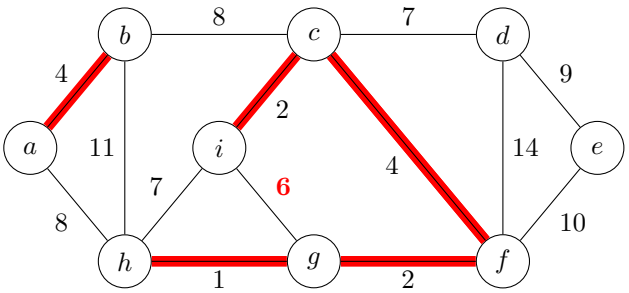
(c) Etape 3 : (g,f) de coût minimal et ne forme pas un cycle



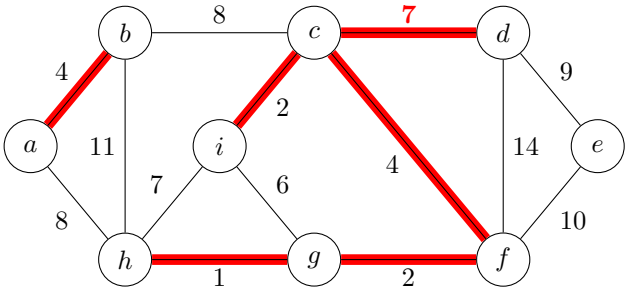
(d) Etape 4 : (a,b) de coût minimal et ne forme pas un cycle



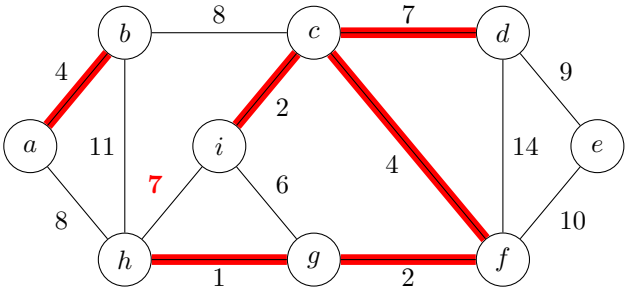
(e) Etape 5 : (c,f) de coût minimal et ne forme pas un cycle



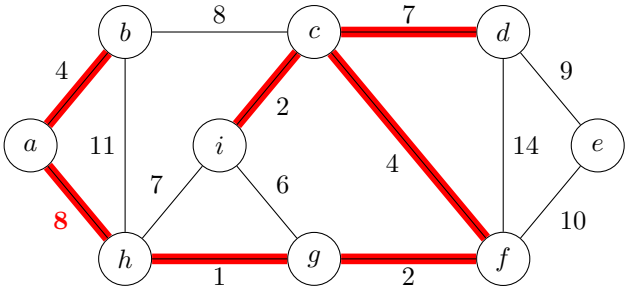
(f) Etape 6 : (i,g) de coût minimal et forme un cycle



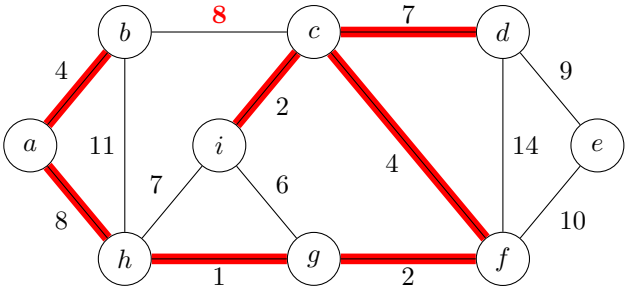
(g) Etape 7 : (c,d) de coût minimal et ne forme pas un cycle



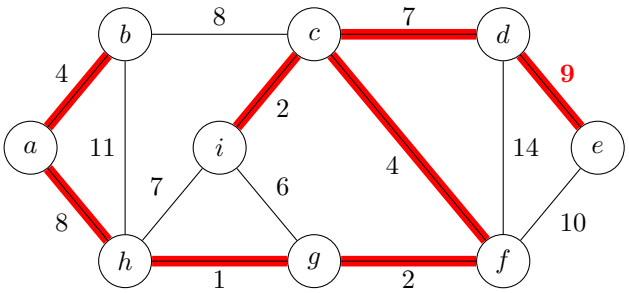
(h) Etape 8 : (h,i) de coût minimal et forme un cycle



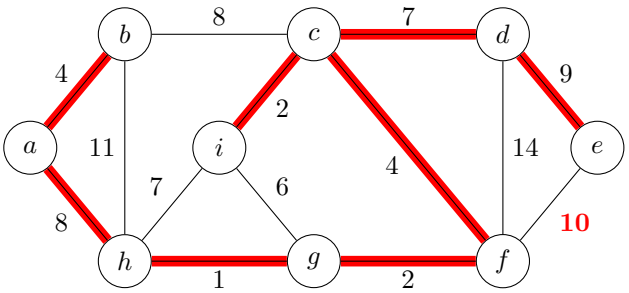
(i) Etape 9 : (a,h) de coût minimal et ne forme pas un cycle



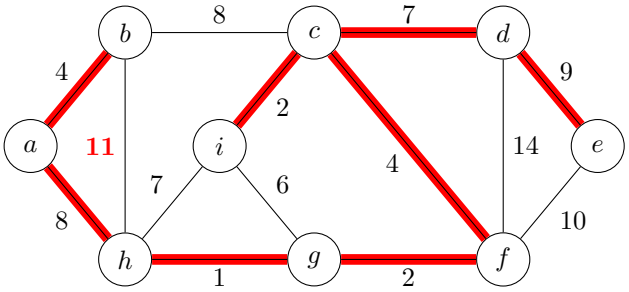
(j) Etape 10 : (b,c) de coût minimal et forme un cycle



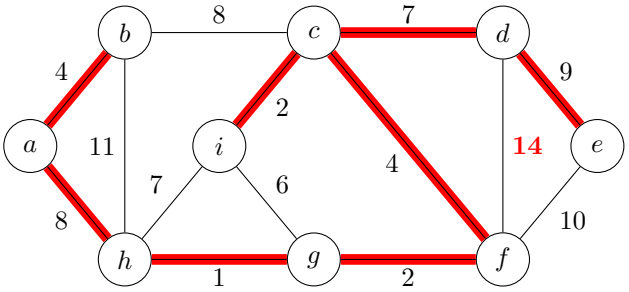
(k) Etape 11 : (d,e) de coût minimal et ne forme pas un cycle



(l) Etape 12 : (f,e) de coût minimal et forme un cycle



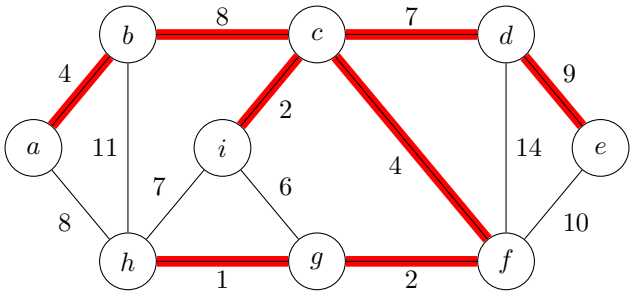
(m) Etape 13 : (b,h) de coût minimal et forme un cycle



(n) Etape 14 : (d,f) de coût minimal et forme un cycle

La figure donnée à l'étape 14 fournit un arbre couvrant minimal du graphe.

Remarque 2.4 Lorsqu'il existe plusieurs arêtes de même coût minimal, le choix d'une arête ou d'une autre peut donner des arbres couvrants minimaux différents. Par exemple, le choix de l'arête (b,c) au lieu de l'arête (a,h) dans l'étape 9, puis le choix de l'arête (a,h) dans l'étape 10 fournit l'arbre couvrant minimal :



qui est différent de celui obtenu précédemment.

2.4 Algorithme de Prim

◅ **Donnée de l'algorithme.**

- un graphe connexe non orienté $G = (S,A)$, valué et simple.

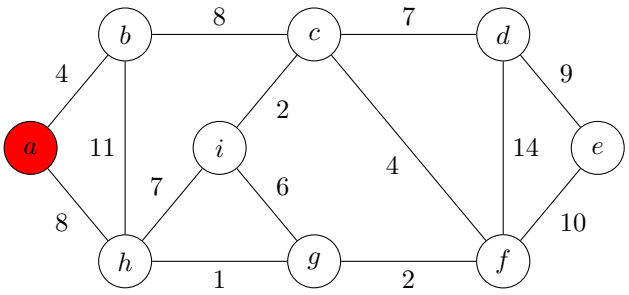
◅ **Idée générale de l'algorithme.**

L'algorithme de Prim est basé sur le fait qu'un arbre à n sommets a $n-1$ *arêtes*. Son principe est le suivant :

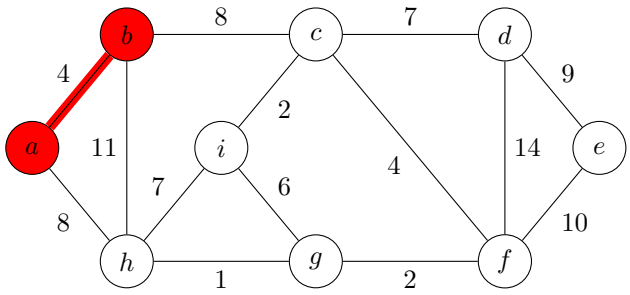
- Choisir arbitrairement un sommet du graphe et faire croître un arbre à partir de celui-ci en ajoutant à l'arbre déjà construit une arête permettant de relier un sommet du graphe non déjà présent dans l'arbre.
- Chaque augmentation se fait de la manière la plus économique possible (choix de l'arête de coût minimal).
- On s'arrête lorsque tous les sommets ont été atteints.

Avec une bonne implémentation, la complexité de l'algorithme de Kruskal est en $O(\text{card}(A) + \text{card}(S)\log_2(\text{card}(S)))$.

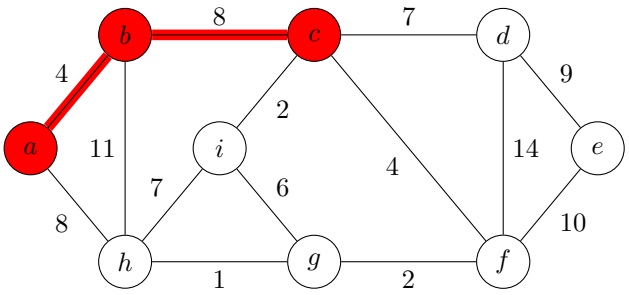
◅ **Un exemple pas à pas.** Les figures ci-dessous illustrent l'exécution de l'algorithme de Prim sur le graphe de l'exemple 2.1.



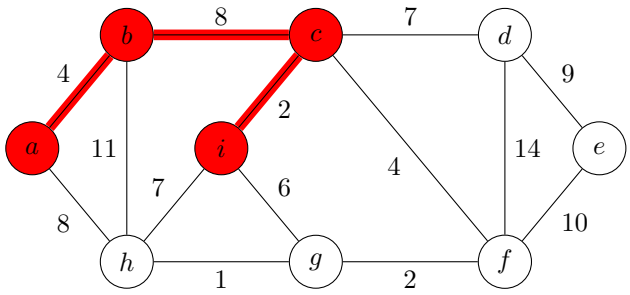
(a) Etape 1 : choix du sommet a comme racine de l'arbre



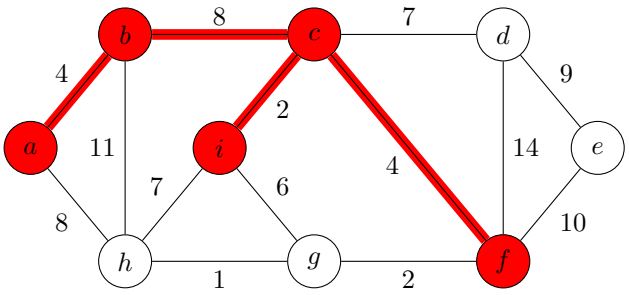
(b) Etape 2 : (a,b) arête de coût minimal issue du sommet a



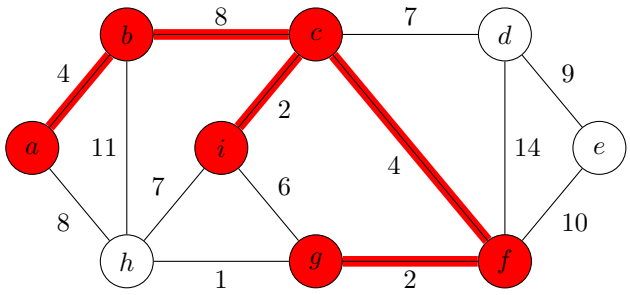
(c) Etape 3 : (b,c) arête de coût minimal issue des sommets a et b



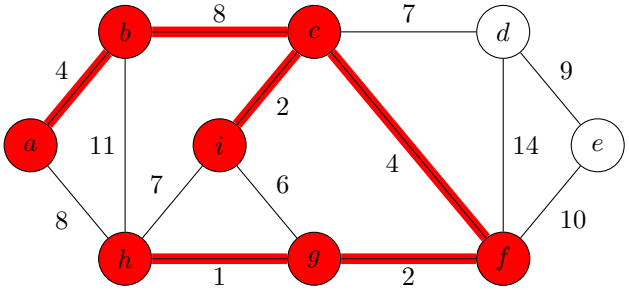
(d) Etape 4 : (c,i) arête de coût minimal issue des sommets a, b et c



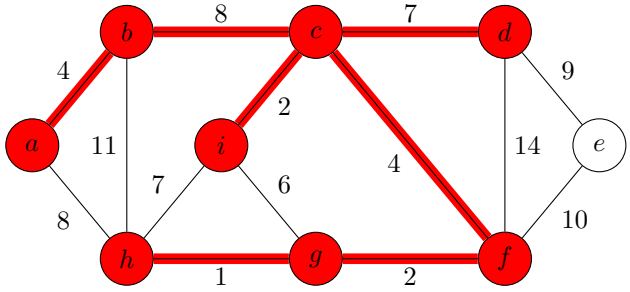
(e) Etape 5 : (c,f) arête de coût minimal issue des sommets a, b, c et i



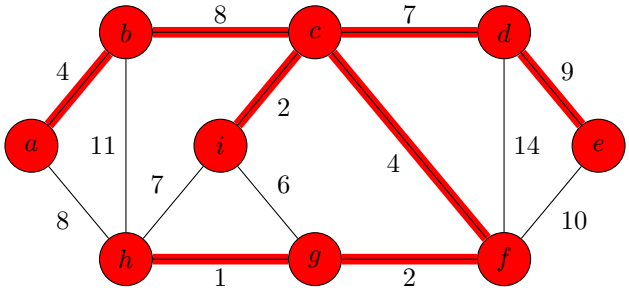
(f) Etape 6 : (f,g) arête de coût minimal issue des sommets a, b, c, i et f



(g) Etape 7 : (c,f) arête de coût minimal issue des sommets a, b, c, i, f et g



(h) Etape 8 : (c,d) arête de coût minimal issue des sommets a, b, c, i, f, g et h

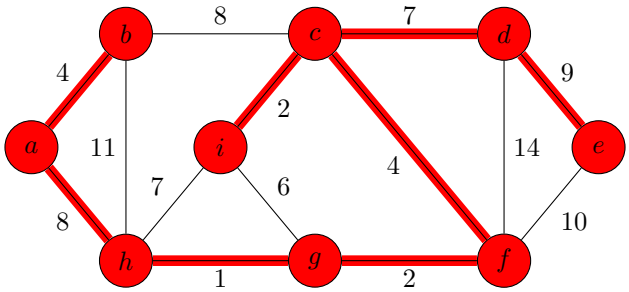


(i) Etape 9 : (d,e) arête de coût minimal issue des sommets a, b, c, i, f, g, h et d

La figure donnée à l'étape 9 fournit un arbre couvrant minimal du graphe.

Remarque 2.5

- Les deux algorithmes de Kruskal et Prim peuvent fournir des arbres couvrants minimaux identiques ou non.
- Comme pour l'algorithme de Kruskal, dans le cas où il existe plusieurs arêtes de même coût minimal, le choix d'une arête ou d'une autre peut donner des arbres couvrants minimaux différents. Par exemple, le choix de l'arête (a,h) au lieu de l'arête (b,c) dans l'étape 3 fournit l'arbre couvrant minimal :



qui est différent de celui obtenu précédemment, mais qui est le même que celui obtenu initialement dans l'algorithme de Kruskal.